Brian Miller
Cobb Peterson
Anthony Wash

ECE3663 Design Project:
Design Review #2

Progress:

We have made significant process in our design since the last report and have fully implemented the adder, subtractor, shifter, register, and the top level connections to make the circuit work properly. Although the circuit has not been fully optimized, we still have implemented a fully-functional and efficient design.

Justification for Design Decisions:

16b-Adder->

We used a ripple carry adder built with mirror adder blocks to implement the 16b adder. To design the circuit, we first built a 1b mirror adder, connected two of them and added necessary inverters to created a 2b mirror adder, then connected eight 2b mirror adders to create the 16b mirror adder.

We designed the 1b mirror adder with the topology provided in class then and connected the carry-in bit of the second mirror adder to the carry-out bit of the first mirror adder. To make sure the output was not inverted, we inverted the sum-bit of the first mirror adder and the two input bits of the second mirror adder.

We chose to use this topology because of the its ease of design and efficiency. Instead of using the first 1b adder described in class, which uses a significant amount of area and leads to large delay times, we used the mirror adder.

16b-Subtractor->

Our original decision was to implement the subtractor as a separate block. We then determined that we could save an inordinate amount of area on our chip if we implemented the subtractor and adder as the same block. To do this, we use the two's complement concept where input B has its bits inverted and then 1 added to it. When this new input B is added to input A, the result is a subtraction with input B being subtracted from input A.

To implement this design, we used a 2-input xnor gate at each of the B inputs. The two bits passed to the xnor are a select bit and the original B bit. If the select bit is 1, the original bit is passed through and the ALU implements the ADD function. If the select bit is 0, the original bit is inverted and passed into the ALU, which implements the SUBTRACT function. The carry-in bit of the adder is fed by the an inverter with the select bit as input. This makes the carry-in bit equal to 0 in an add operation and 1 in a subtract operation, which we need for two's complement.

We chose to use this implementation because we could use the previous adder construction and only have to add a few more logic gates. This allowed us to significantly

reduce area compared to having two separate blocks. This implementation will allow for easier optimizations in the future because any optimizations we add to our adder will also benefit the subtractor.

16b-Shifter->

In implementing the shifter, we considered different types of shifters to build. We considered whether to have the most significant shifted bits shifted to the lower bits that had been shifted (barrel shifter) or having the most significant shifted bits “falling off” the shifter and zeros being loaded into the least significant bits. We decided to implement the shifter where the least significant bits were loaded with zeros and the most significant shifted bits “fall off” the shifter. We chose this implementation because it allows us to easily have a shift operation that can be used for base-2 multiplication and addition.

To implement the shifter, we designed a one bit shifter block that routed the provided input to one of four outputs based on the two provided select lines. The four outputs of the shifter block are connected to the four different output locations representing shifts of one to four bits. Our two select lines b1 and b0 determined the 4 different possible shifts and the results of their inputs are below.

b1=Vdd,b0=Vdd -> 4-bit shift
b1=Vdd,b0=Vss -> 3-bit shift
b1=Vss,b0=Vdd -> 2-bit shift
b1=Vss,b0=Vss -> 1-bit shift

We have 19 shifter blocks in our current circuit. 15 are used for the input bits of the circuit, we do not include a shifter block for the most significant bit because any bit shift of this bit will force this bit to “fall off” the shifter. The other 4 shifter blocks are not connected to the the inputs of the circuit but hold the input value of 0. Their outputs are linked to the least significant output bits of the circuit and load zeros into the shifter.

16-b-Register->

For our register design, we chose to implement the same basic register that was presented in Problem Set 7, consisting of a master and slave latch. Each latch consists of two tri-state inverters and an inverter, where the master and slave are set to operate on alternating clock values (i.e. master latch is transparent on low, slave is transparent on high). This gave us a rising edge triggered register. Using 16 instances of this register in parallel, we were able to create a complete 16-bit register.

Remaining Tasks:

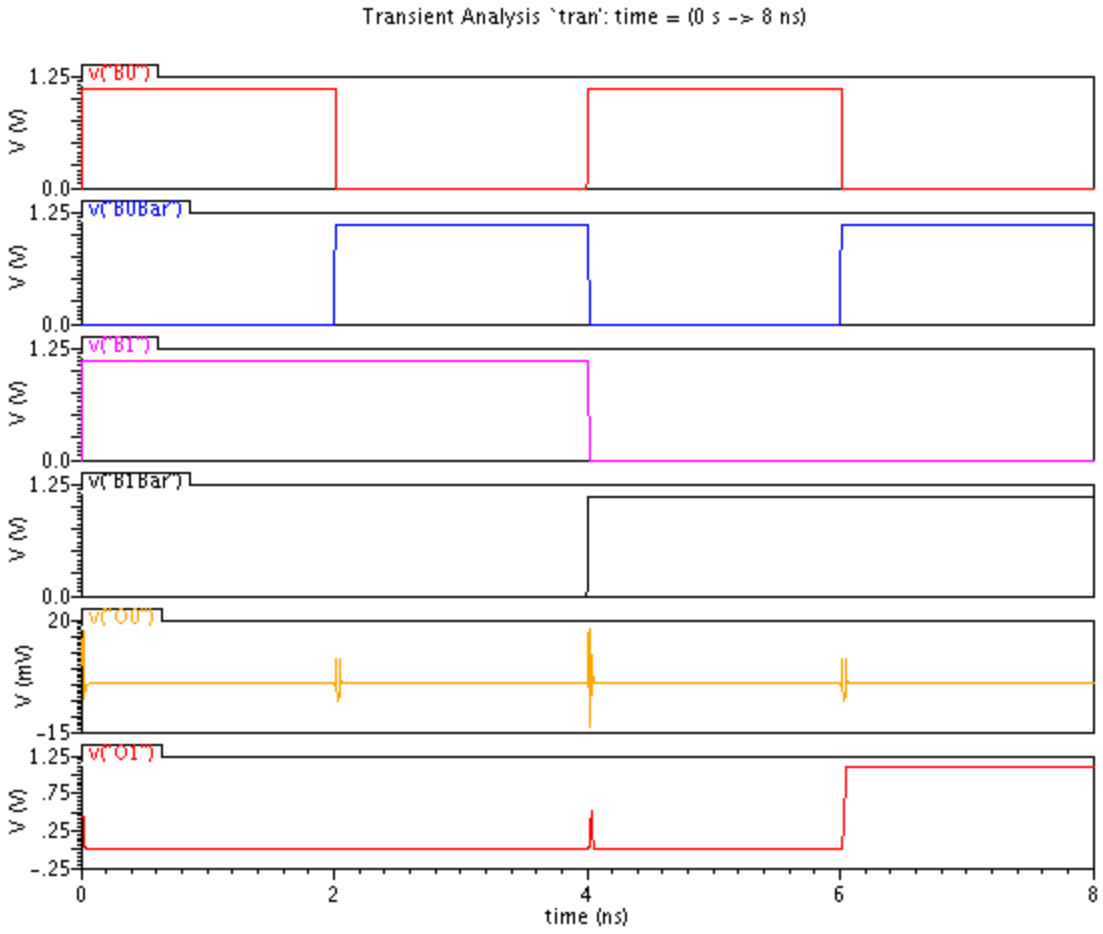
We have designed a fully functional ALU that has low delay and area costs, but our new goal is to significantly optimize these parameters and improve our circuit design. The goal is to have a robust circuit that works extremely well and will always return correct values with optimized metrics.

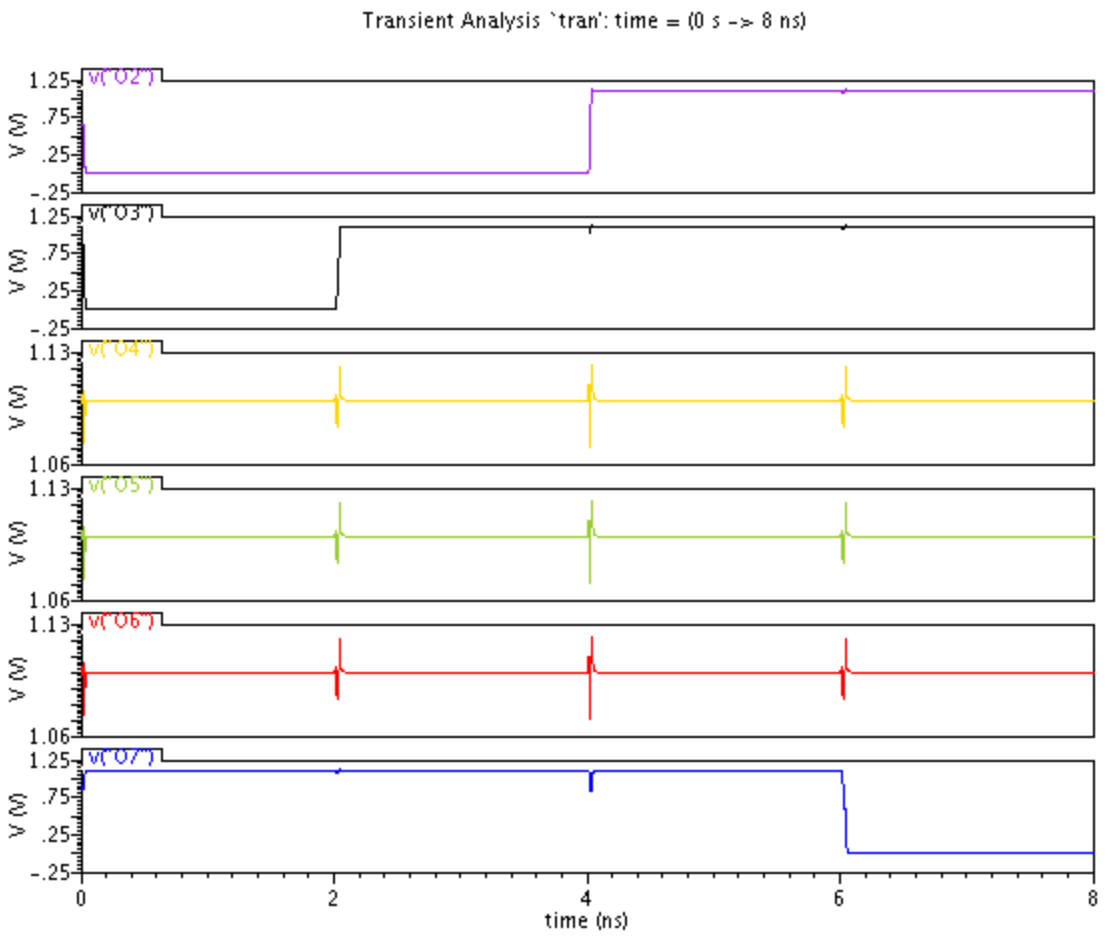
Extra Function:

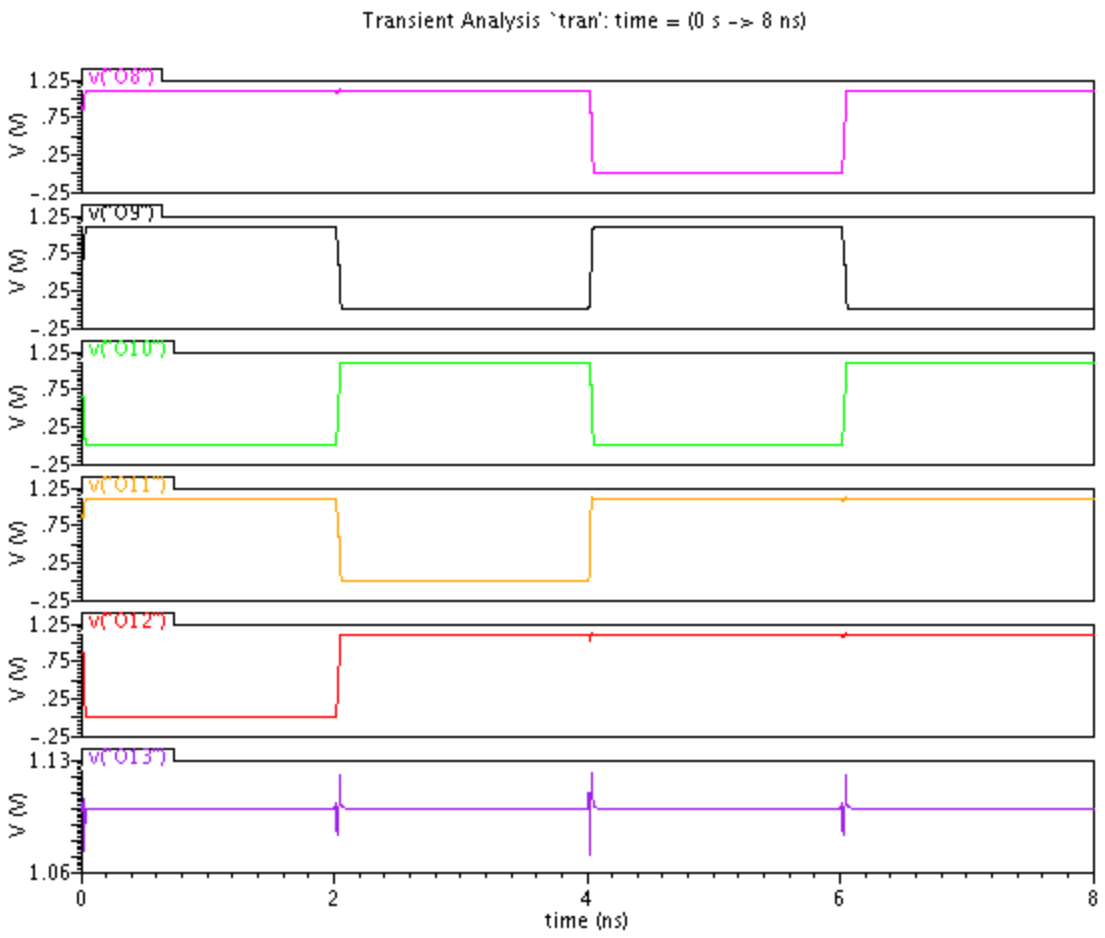
We are going to implement a multiplier for our extra function. We have been discussing different possible implementations and our current decision is to implement the multiplier using the Wallace Tree Multiplier implementation discussed in class.

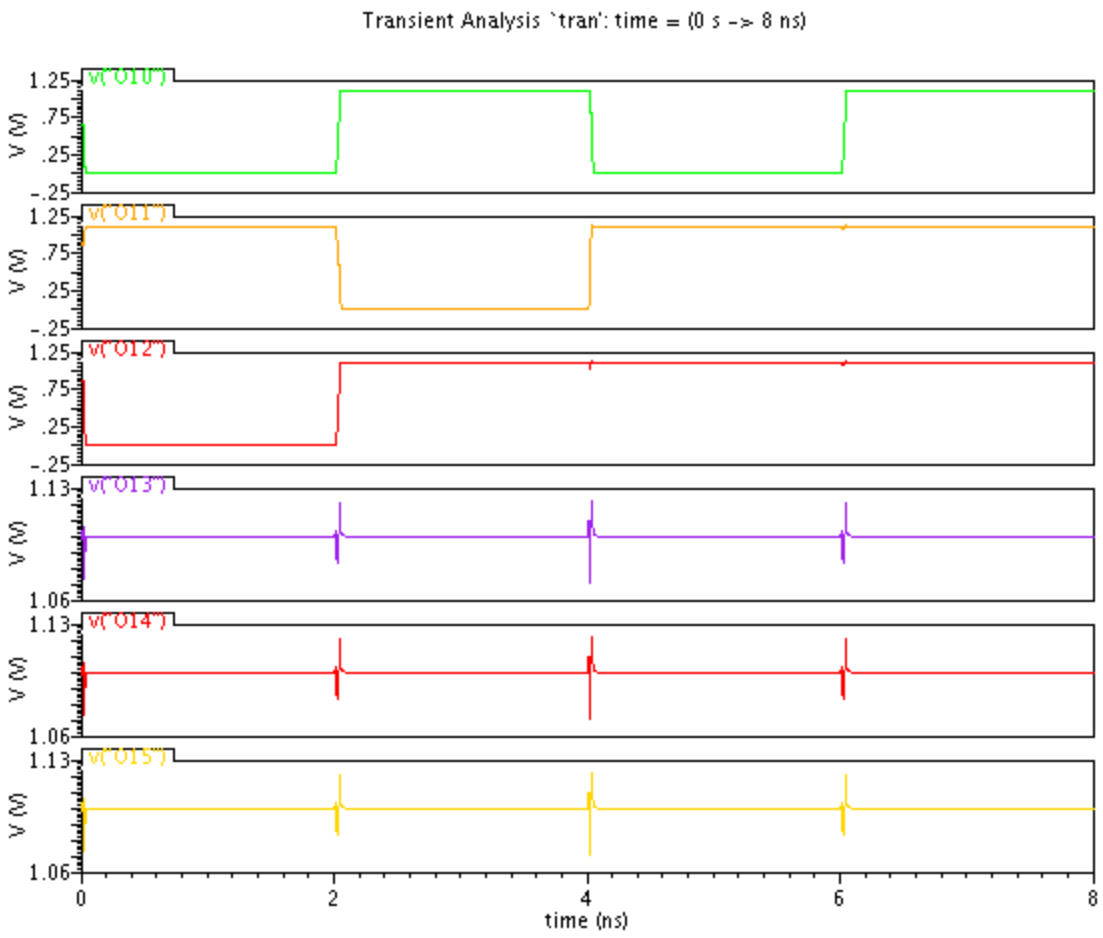
Shifter Simulation

B0, B0Bar, B1, and B1Bar are the shift select inputs. O0,O1,...,O15 are the shifter outputs. For this simulation, input A=1111111010111111. For each combination of select bits, the two zeros in A are shifted the appropriate number of bits. Zeros are shifted into the least significant bit if a one is shifted away.



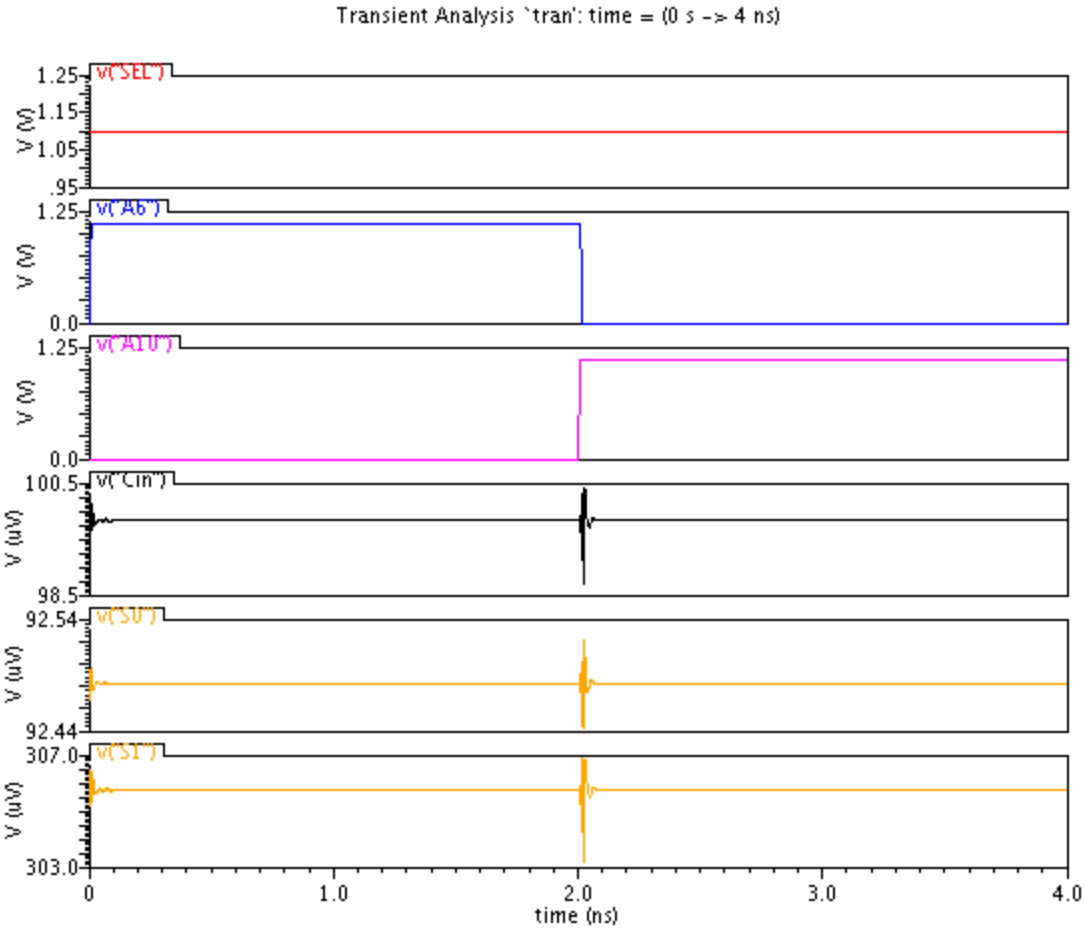

Transient Analysis `tran': time = (0 s -> 8 ns)
v("O2")
v("O3")
v("O4")
v("O5")
v("O6")
v("O7")
V (V)
1.25
.75
.25
-.25
1.13
1.06
0
2
4
6
8
time (ns)

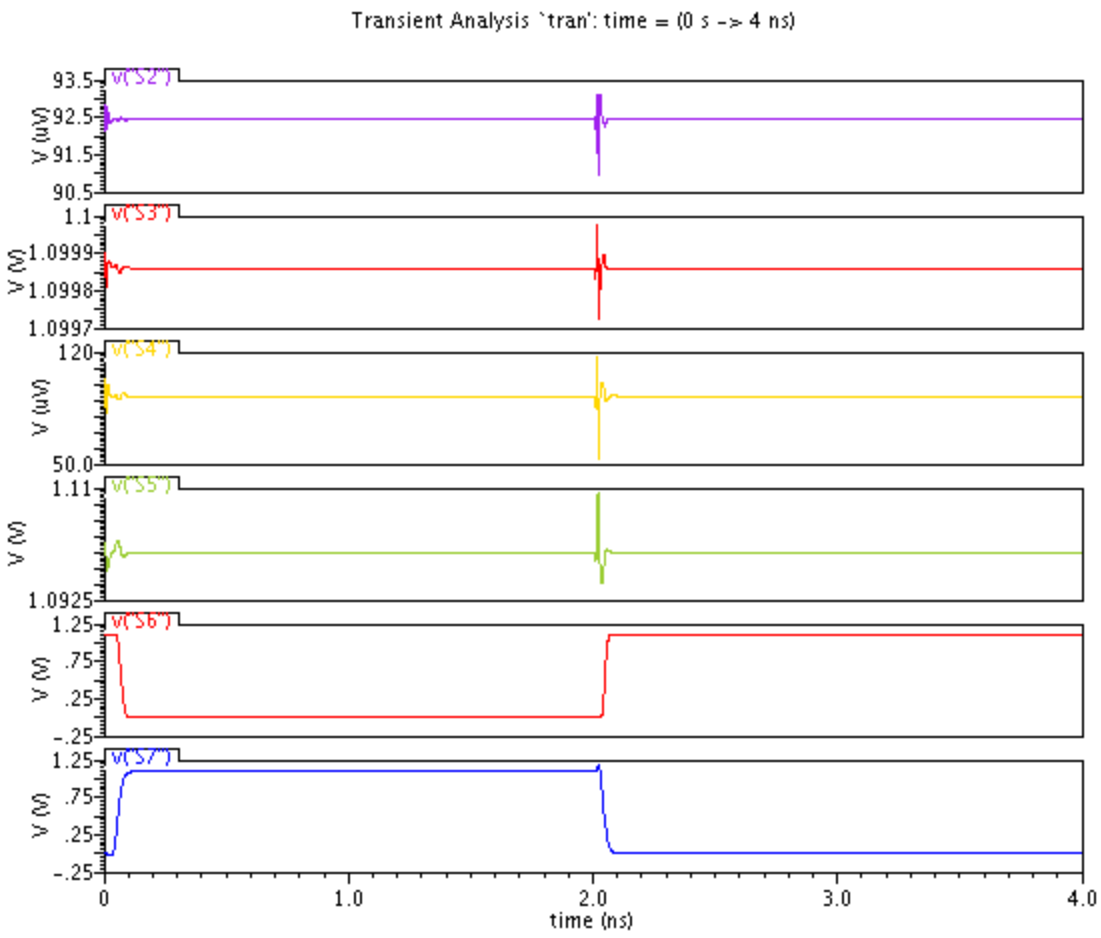

Transient Analysis `tran': time = (0 s -> 8 ns)
v("O8")
v("O9")
v("O10")
v("O11")
v("O12")
v("O13")
1.25
.75
.25
-.25
1.13
1.06
V (V)
0
2
4
6
8
time (ns)

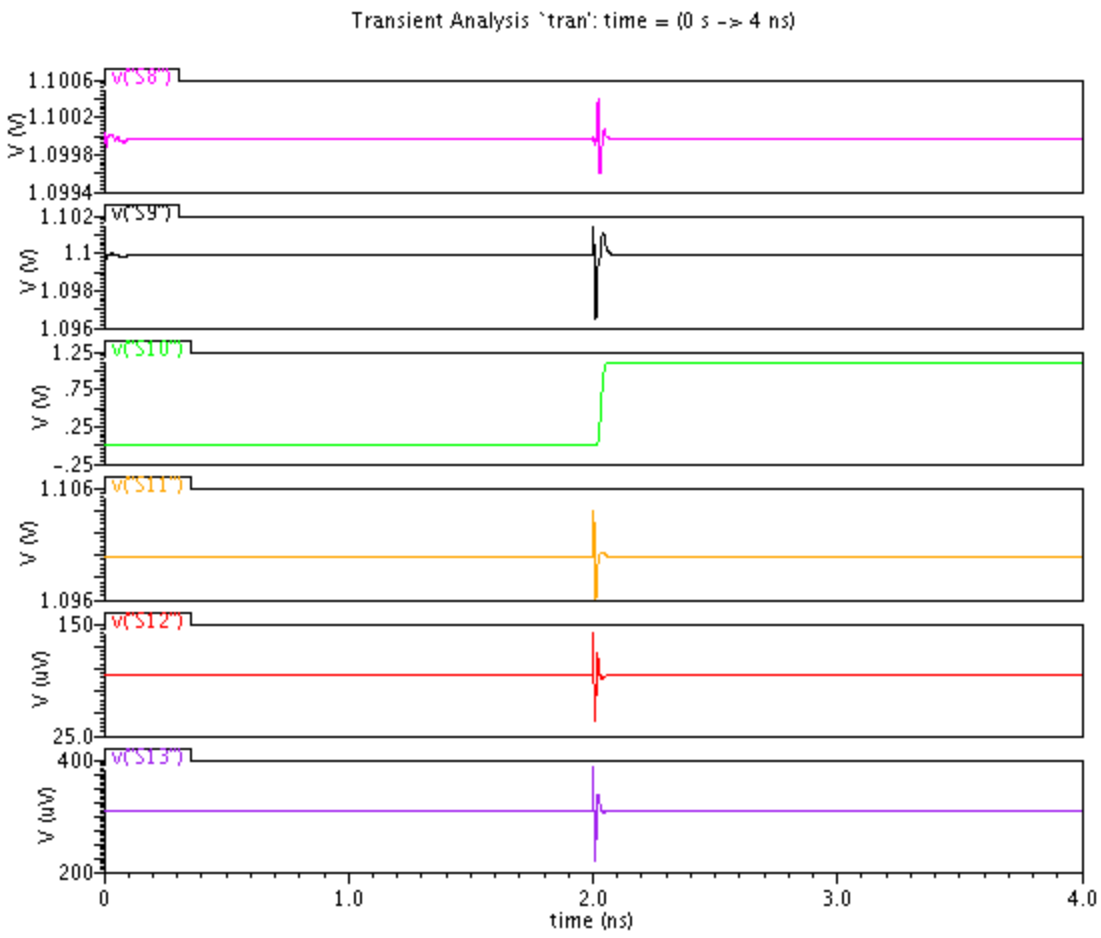

Transient Analysis `tran': time = (0 s -> 8 ns)
v("OI0")
v("OI1")
v("OI2")
v("OI3")
v("OI4")
v("OI5")
1.25
.75
.25
-.25
1.13
1.06
V (V)
0
2
4
6
8
time (ns)

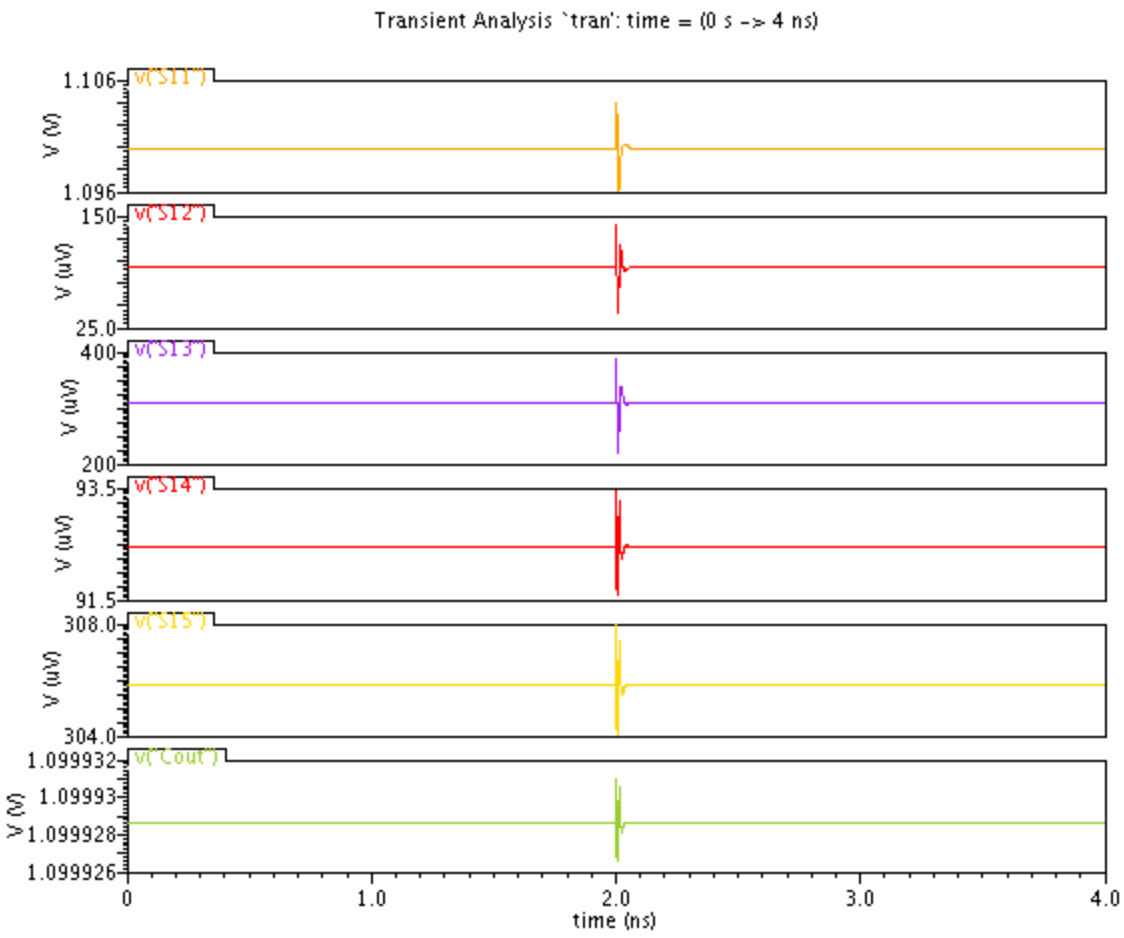
Adder Simulation

The select line is high, which selects the ADD operation. Initially, A=0111100101100001 and B=1001001001000111. Bits 6 and 10 are toggled to test two different addition operations. Chronologically, the two sums are(including the carry out): S0=10000101110101000 and S1=10000111101101000. S0,S1,...,S15 are the sum bit plots.



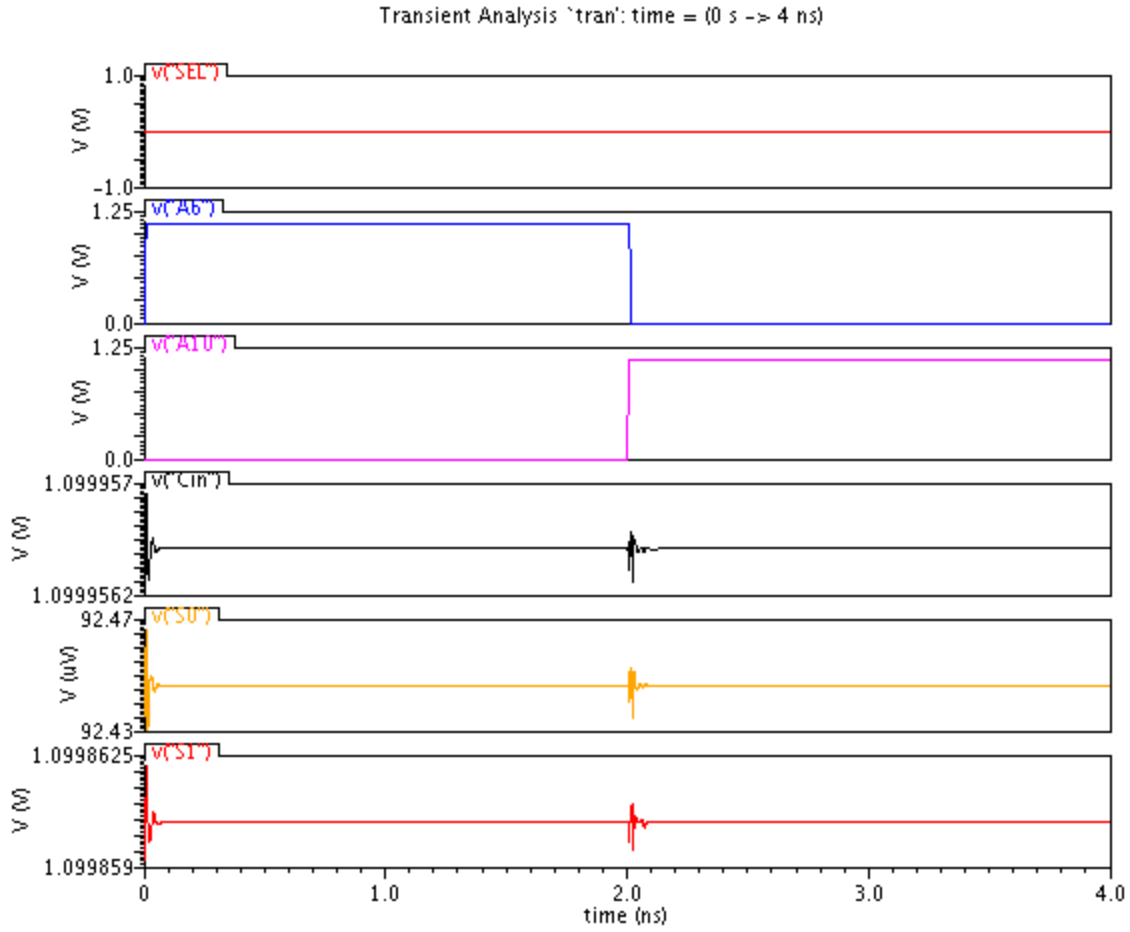

Transient Analysis `tran': time = (0 s -> 4 ns)
v("S2")
93.5
92.5
91.5
90.5
V (V)
v("S3")
1.1
1.0999
1.0998
1.0997
V (V)
v("S4")
120
50.0
V (V)
v("S5")
1.11
1.0925
V (V)
v("S6")
1.25
.75
.25
-.25
V (V)
v("S7")
1.25
.75
.25
-.25
V (V)
0
1.0
2.0
3.0
4.0
time (ns)

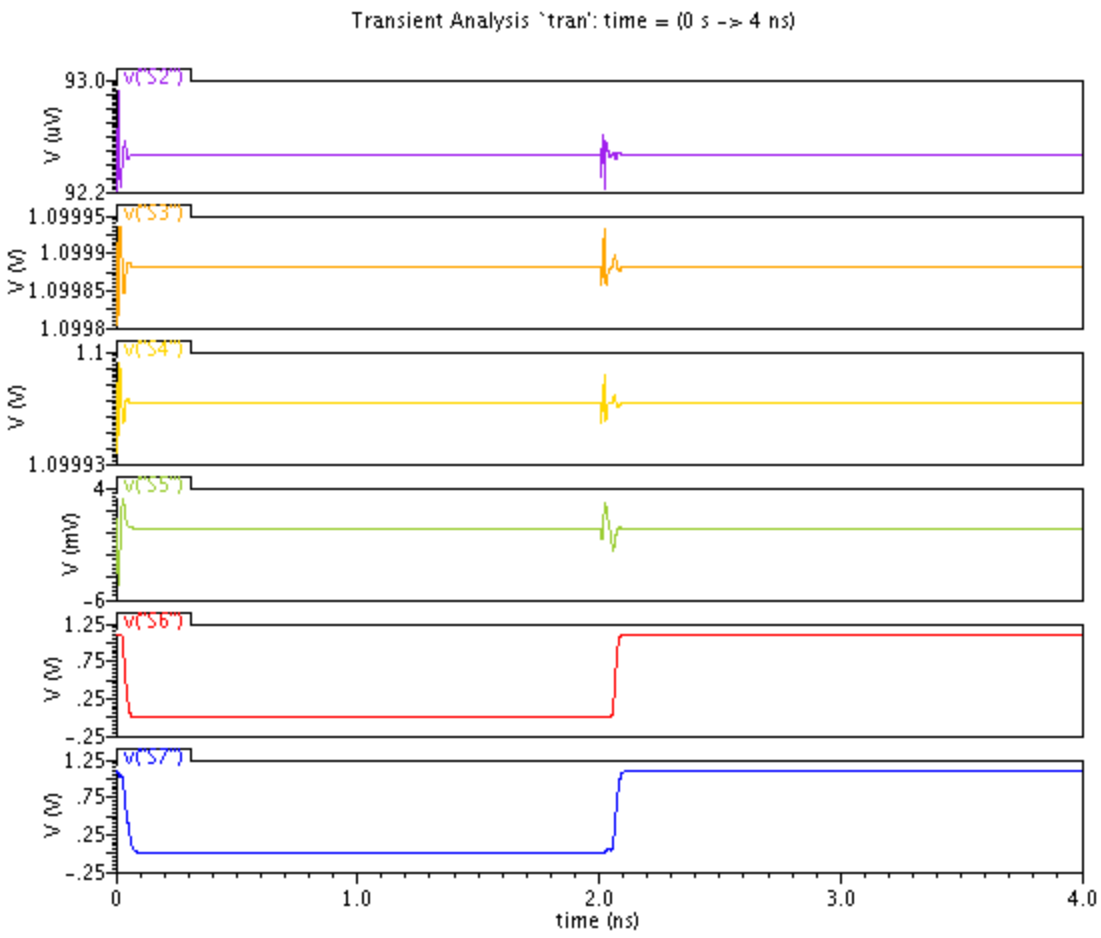

Transient Analysis `tran': time = (0 s -> 4 ns)
v("S8")
1.1006
1.1002
1.0998
1.0994
V (V)
v("S9")
1.102
1.1
1.098
1.096
V (V)
v("S10")
1.25
.75
.25
-.25
V (V)
v("S11")
1.106
1.096
V (V)
v("S12")
150
25.0
V (mV)
v("S13")
400
200
V (mV)
0
1.0
2.0
3.0
4.0
time (ns)

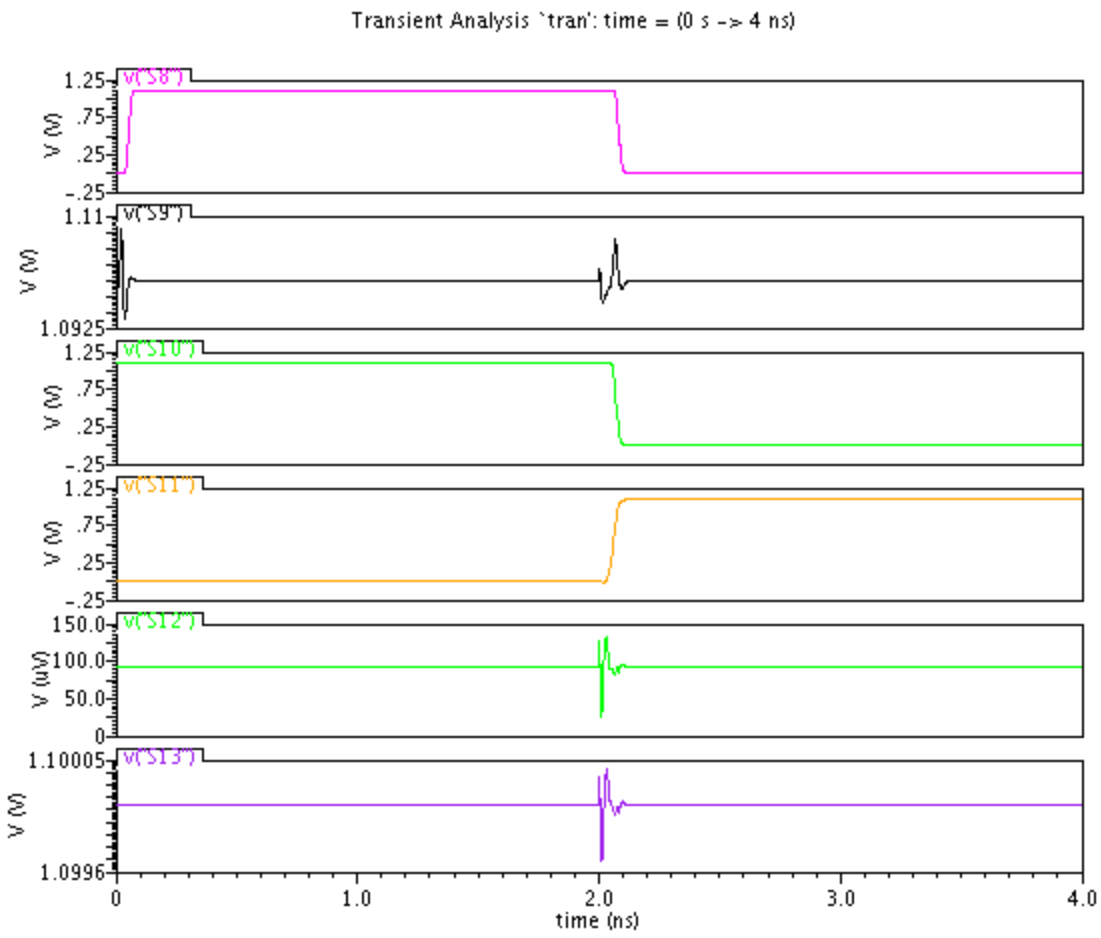

Transient Analysis `tran': time = (0 s -> 4 ns)
v("SI1")
1.106
1.096
V (V)
v("SI2")
150
25.0
V (mV)
v("SI3")
400
200
V (mV)
v("SI4")
93.5
91.5
V (mV)
v("SI5")
308.0
304.0
V (mV)
v("Cout")
1.099932
1.09993
1.099928
1.099926
V (V)
0
1.0
2.0
3.0
4.0
time (ns)

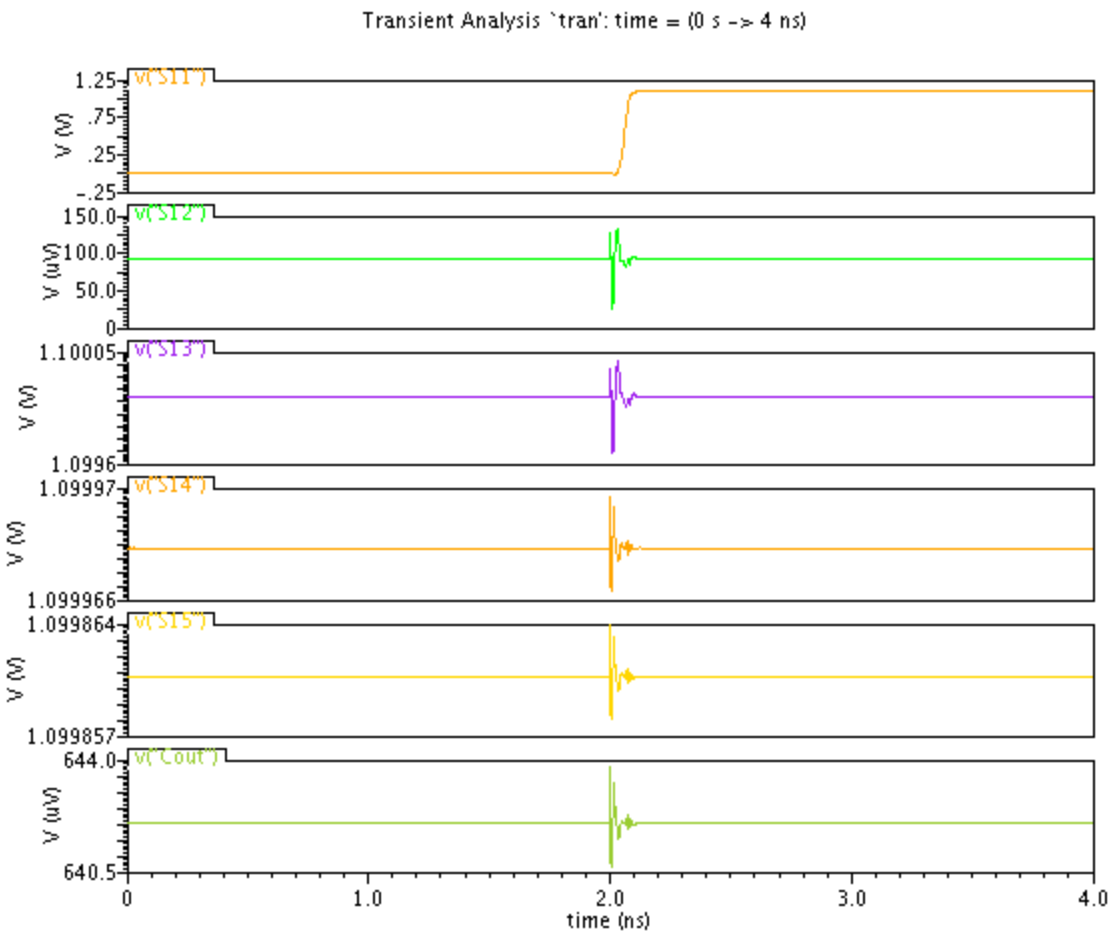
Subtraction Simulation

The select line is low, which selects the SUB operation. Initially, A=0111100101100001 and B=1001001001000111. Because we are performing subtraction using two's complement, input B will be inverted and incremented by one before being passed to the adder. Thus, the number being added to A will be: B=0110110110111001. Bits 6 and 10 of input A are toggled to test two different subtraction operations. Chronologically, the two sums are(including the carry out): S0=0111001110001101O and S1=01110101011011010. S0,S1,...,S15 are the sum bit plots.



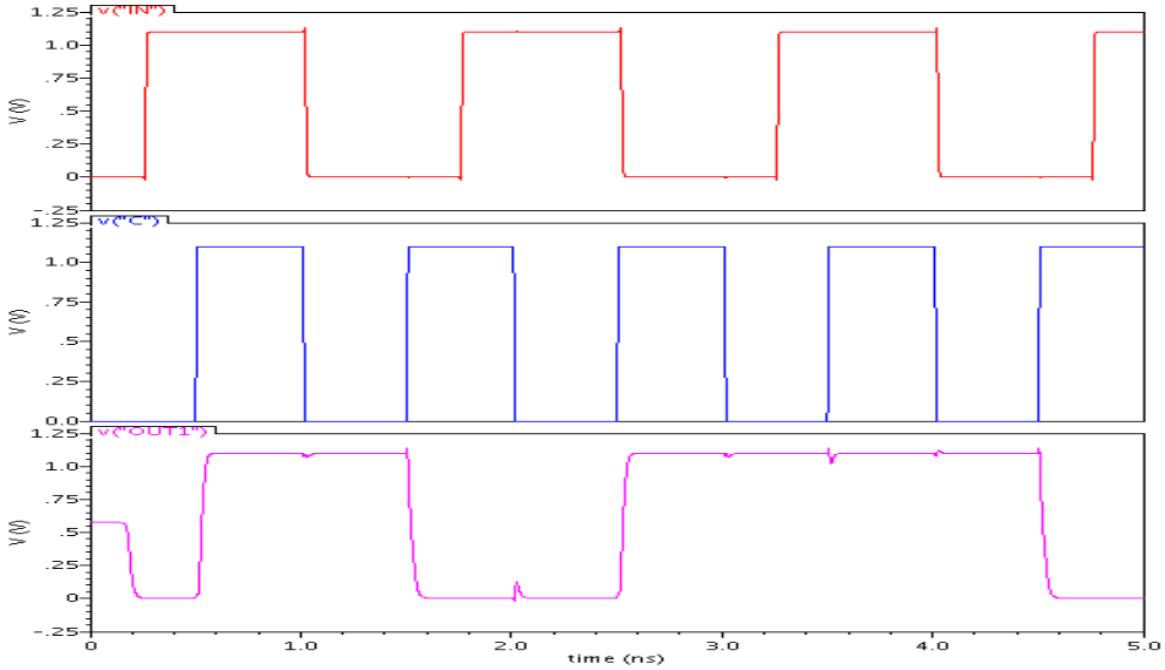

Transient Analysis `tran': time = (0 s -> 4 ns)
V("S2")
93.0
92.2
V (uV)
V("S3")
1.09995
1.0999
1.09985
1.0998
V (V)
V("S4")
1.1
1.09993
V (V)
V("S5")
4
-6
V (mV)
V("S6")
1.25
.75
.25
-.25
V (V)
V("S7")
1.25
.75
.25
-.25
V (V)
0
1.0
2.0
3.0
4.0
time (ns)


Transient Analysis `tran': time = (0 s -> 4 ns)
V("S8")
V("S9")
V("S10")
V("S11")
V("S12")
V("S13")
V (V)
V (mV)
1.25
.75
.25
-.25
1.11
1.0925
150.0
100.0
50.0
0
1.10005
1.0996
0
1.0
2.0
3.0
4.0
time (ns)


Transient Analysis `tran': time = (0 s -> 4 ns)
v("SI1")
1.25
.75
.25
-.25
V (V)
v("SI2")
150.0
100.0
50.0
0
V (mV)
v("SI3")
1.10005
1.0996
V (V)
v("SI4")
1.09997
1.099966
V (V)
v("SI5")
1.099864
1.099857
V (V)
v("Cout")
644.0
640.5
V (mV)
0
1.0
2.0
3.0
4.0
time (ns)

Register Simulation

The following simulation shows the operation of a single register with an alternating input (shown as v(“IN”)). As this input changes, the clock (shown as v(“C”)) alternates as well, capturing and storing the input value at each rising edge. OUT1 is the output for this register. The complete 16-Bit Register uses 16 of these subcircuits in parallel to complete its task.



Operation	Worst-Case Delay	Scenario
ADD	188.78ps	**B=0111111111111111** **A=0000000000000000** Bit 0 of A is toggled high, causing a ripple through the carry chain to change sum bit 15 from 0 to 1.
SUB	208.78ps	**B=1000000000000001** Since B is inverted and then incremented by one before it is passed to the adder, B’+1=0111111111111111 and the adder is effectively summing A and B’+1. **A=0000000000000000** Bit 0 of A is toggled high, causing a ripple through the carry chain to change sum bit 15 from 0 to 1.
SHIFT	28.7ps	Any input being shifted can be shifted any amount to see the worst case delay i.e. the signal is only ever delayed by the control logi

		and one transmission gate.
AND	15.91ps	This is the delay of the AND gate with FO2 inverter load.
OR	17.20ps	This is the delay of the OR gate with FO2 inverter load.
REGISTER	22.06ps	This is the clock-to-output delay of the register.
PASS	1ps	The worst-case delay is just the propagation delay of one transmission gate.

Netlists

ADD/SUB

```
subckt MirrorAdder VDD VSS A B Ci CoBar SBar
M20 (SBar Ci net9 net9) PMOS_VTL w=540.0n l=50n as=5.67e-14 ad=5.67e-14 \
        ps=750.0n pd=750.0n ld=105n ls=105n m=1
M19 (net9 B net13 net13) PMOS_VTL w=540.0n l=50n as=5.67e-14 ad=5.67e-14 \
        ps=750.0n pd=750.0n ld=105n ls=105n m=1
M18 (net13 A VDD VDD) PMOS_VTL w=540.0n l=50n as=5.67e-14 ad=5.67e-14 \
        ps=750.0n pd=750.0n ld=105n ls=105n m=1
M17 (net24 Ci VDD VDD) PMOS_VTL w=360.0n l=50n as=3.78e-14 ad=3.78e-14 \
        ps=570.0n pd=570.0n ld=105n ls=105n m=1
M16 (net24 A VDD VDD) PMOS_VTL w=360.0n l=50n as=3.78e-14 ad=3.78e-14 \
        ps=570.0n pd=570.0n ld=105n ls=105n m=1
M15 (net24 B VDD VDD) PMOS_VTL w=360.0n l=50n as=3.78e-14 ad=3.78e-14 \
        ps=570.0n pd=570.0n ld=105n ls=105n m=1
M14 (SBar CoBar net24 net24) PMOS_VTL w=360.0n l=50n as=3.78e-14 \
        ad=3.78e-14 ps=570.0n pd=570.0n ld=105n ls=105n m=1
M4 (CoBar Ci net52 net52) PMOS_VTL w=1.08u l=50n as=1.134e-13 ad=1.134e-13 \
        ps=1.29u pd=1.29u ld=105n ls=105n m=1
M3 (CoBar A net41 net41) PMOS_VTL w=360.0n l=50n as=3.78e-14 ad=3.78e-14 \
        ps=570.0n pd=570.0n ld=105n ls=105n m=1
M2 (net41 B VDD VDD) PMOS_VTL w=360.0n l=50n as=3.78e-14 ad=3.78e-14 \
        ps=570.0n pd=570.0n ld=105n ls=105n m=1
M1 (net52 B VDD VDD) PMOS_VTL w=1.08u l=50n as=1.134e-13 ad=1.134e-13 \
        ps=1.29u pd=1.29u ld=105n ls=105n m=1
M0 (net52 A VDD VDD) PMOS_VTL w=1.08u l=50n as=1.134e-13 ad=1.134e-13 \
        ps=1.29u pd=1.29u ld=105n ls=105n m=1
```

```
M23 (net56 B VSS VSS) NMOS_VTL w=270.0n l=50n as=2.835e-14 ad=2.835e-14 \
        ps=480.0n pd=480.0n ld=105n ls=105n m=1
M22 (net60 A net56 net56) NMOS_VTL w=270.0n l=50n as=2.835e-14 \
        ad=2.835e-14 ps=480.0n pd=480.0n ld=105n ls=105n m=1
M21 (SBar Ci net60 net60) NMOS_VTL w=270.0n l=50n as=2.835e-14 \
        ad=2.835e-14 ps=480.0n pd=480.0n ld=105n ls=105n m=1
M13 (SBar CoBar net80 net80) NMOS_VTL w=180.0n l=50n as=1.89e-14 \
        ad=1.89e-14 ps=390.0n pd=390.0n ld=105n ls=105n m=1
M12 (net80 Ci VSS VSS) NMOS_VTL w=180.0n l=50n as=1.89e-14 ad=1.89e-14 \
        ps=390.0n pd=390.0n ld=105n ls=105n m=1
M11 (net80 B VSS VSS) NMOS_VTL w=180.0n l=50n as=1.89e-14 ad=1.89e-14 \
        ps=390.0n pd=390.0n ld=105n ls=105n m=1
M10 (net80 A VSS VSS) NMOS_VTL w=180.0n l=50n as=1.89e-14 ad=1.89e-14 \
        ps=390.0n pd=390.0n ld=105n ls=105n m=1
M9 (CoBar A net88 net88) NMOS_VTL w=180.0n l=50n as=1.89e-14 ad=1.89e-14 \
        ps=390.0n pd=390.0n ld=105n ls=105n m=1
M8 (net88 B VSS VSS) NMOS_VTL w=180.0n l=50n as=1.89e-14 ad=1.89e-14 \
        ps=390.0n pd=390.0n ld=105n ls=105n m=1
M7 (net96 B VSS VSS) NMOS_VTL w=540.0n l=50n as=5.67e-14 ad=5.67e-14 \
        ps=750.0n pd=750.0n ld=105n ls=105n m=1
M6 (net96 A VSS VSS) NMOS_VTL w=540.0n l=50n as=5.67e-14 ad=5.67e-14 \
        ps=750.0n pd=750.0n ld=105n ls=105n m=1
M5 (CoBar Ci net96 net96) NMOS_VTL w=540.0n l=50n as=5.67e-14 ad=5.67e-14 \
        ps=750.0n pd=750.0n ld=105n ls=105n m=1
ends MirrorAdder

//2bit Mirror adder block
subckt MirrorAdder_2bit VDD VSS A0 A1 B0 B1 Ci Co S0 S1
INV1 (VDD VSS A1 n1) ece3663Inverter
INV2 (VDD VSS B1 n2) ece3663Inverter
INV3 (VDD VSS n4 S0) ece3663Inverter
MA1 (VDD VSS A0 B0 Ci n3 n4) MirrorAdder
MA2 (VDD VSS n1 n2 n3 Co S1) MirrorAdder
ends MirrorAdder_2bit

//8bit Mirror Adder block
subckt MirrorAdder_8bit VDD VSS A0 A1 A2 A3 A4 A5 A6 A7 B0 B1 B2 B3 B4 B5 B6 B7 S0 S1
S2 S3 S4 S5 S6 S7 Ci Co
2bMA1 (VDD VSS A0 A1 B0 B1 Ci n2 S0 S1) MirrorAdder_2bit
2bMA2 (VDD VSS A2 A3 B2 B3 n2 n3 S2 S3) MirrorAdder_2bit
2bMA3 (VDD VSS A4 A5 B4 B5 n3 n4 S4 S5) MirrorAdder_2bit
2bMA4 (VDD VSS A6 A7 B6 B7 n4 Co S6 S7) MirrorAdder_2bit
ends MirrorAdder_8bit
```

SHIFT

```
//1bit shift block
subckt oneBitShift VDD VSS Input B0 B0Bar B1 B1Bar OUT1 OUT2 OUT3 OUT4
AND1 (VDD VSS B1Bar B0Bar And1) ece3663AND2
AND2 (VDD VSS B1Bar B0 And2) ece3663AND2
AND3 (VDD VSS B1 B0Bar And3) ece3663AND2
AND4 (VDD VSS B1 B0 And4) ece3663AND2
TX1 (VDD VSS Input And1 OUT1) ece3663tgate
TX2 (VDD VSS Input And2 OUT2) ece3663tgate
TX3 (VDD VSS Input And3 OUT3) ece3663tgate
TX4 (VDD VSS Input And4 OUT4) ece3663tgate
ends oneBitShift
//end

//16bit shifter
S0 (vdd! 0 A0 B0 B0Bar B1 B1Bar O1 O2 O3 O4) oneBitShift
S1 (vdd! 0 A1 B0 B0Bar B1 B1Bar O2 O3 O4 O5) oneBitShift
S2 (vdd! 0 A2 B0 B0Bar B1 B1Bar O3 O4 O5 O6) oneBitShift
S3 (vdd! 0 A3 B0 B0Bar B1 B1Bar O4 O5 O6 O7) oneBitShift
S4 (vdd! 0 A4 B0 B0Bar B1 B1Bar O5 O6 O7 O8) oneBitShift
S5 (vdd! 0 A5 B0 B0Bar B1 B1Bar O6 O7 O8 O9) oneBitShift
S6 (vdd! 0 A6 B0 B0Bar B1 B1Bar O7 O8 O9 O10) oneBitShift
S7 (vdd! 0 A7 B0 B0Bar B1 B1Bar O8 O9 O10 O11) oneBitShift
S8 (vdd! 0 A8 B0 B0Bar B1 B1Bar O9 O10 O11 O12) oneBitShift
S9 (vdd! 0 A9 B0 B0Bar B1 B1Bar O10 O11 O12 O13) oneBitShift
S10 (vdd! 0 A10 B0 B0Bar B1 B1Bar O11 O12 O13 O14) oneBitShift
S11 (vdd! 0 A11 B0 B0Bar B1 B1Bar O12 O13 O14 O15) oneBitShift
S12 (vdd! 0 A12 B0 B0Bar B1 B1Bar O13 O14 O15 O16) oneBitShift
S13 (vdd! 0 A13 B0 B0Bar B1 B1Bar O14 O15 O16 O17) oneBitShift
S14 (vdd! 0 A14 B0 B0Bar B1 B1Bar O15 O16 O17 O18) oneBitShift
SO1 (vdd! 0 0 B0 B0Bar B1 B1Bar nc0 nc1 nc2 O0) oneBitShift
SO2 (vdd! 0 0 B0 B0Bar B1 B1Bar nc3 nc4 O0 O1) oneBitShift
SO3 (vdd! 0 0 B0 B0Bar B1 B1Bar nc5 O0 O1 O2) oneBitShift
SO4 (vdd! 0 0 B0 B0Bar B1 B1Bar O0 O1 O2 O3) oneBitShift
```

REGISTER

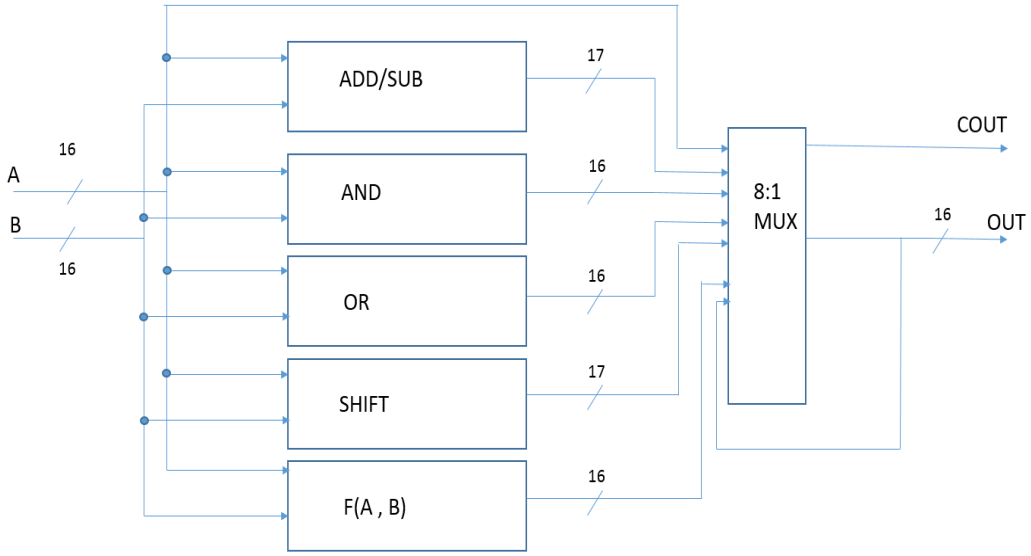
```
subckt ece3663TRISTATE VDD VSS IN C NC OUT
parameters wp=180n wn=90n lp=50n ln=50n mult=1
  M1 (VDD IN net0 VDD) PMOS_VTL w=wp l=lp as=100n*2*wp ad=100n*2*wp ps=200n+2*wp
pd=200n+2*wp m=mult
  M2 (net0 C OUT VDD)  PMOS_VTL w=wp l=lp as=100n*2*wp ad=100n*2*wp ps=200n+2*wp
pd=200n+2*wp m=mult
  M3 (OUT NC net1 VSS) NMOS_VTL w=wn l=ln as=100n*2*wn ad=100n*2*wn ps=200n+2*wn
pd=200n+2*wn m=mult
  M4 (net1 IN VSS VSS) NMOS_VTL w=wn l=ln as=100n*2*wn ad=100n*2*wn ps=200n+2*wn
pd=200n+2*wn m=mult
ends ece3663TRISTATE

subckt ece3663DFLIPFLOP VDD VSS CLK NCLK D Q
  INV_OUT0 (VDD VSS OUT0 OUT1) ece3663Inverter
  INT_OUT1 (VDD VSS Q OUT3) ece3663Inverter
  TRI0 (VDD VSS D CLK NCLK OUT0) ece3663TRISTATE
  TRI1 (VDD VSS OUT1 NCLK CLK OUT0) ece3663TRISTATE
  TRI2 (VDD VSS OUT0 NCLK CLK Q) ece3663TRISTATE
  TRI3 (VDD VSS OUT3 CLK NCLK Q) ece3663TRISTATE
ends ece3663DFLIPFLOP

subckt ece366316B_REGISTER VDD VSS CLK NCLK D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10
D11 D12 D13 D14 D15 Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15
  FF0 (VDD VSS CLK NCLK D0 Q0) ece3663DFLIPFLOP
  FF1 (VDD VSS CLK NCLK D1 Q1) ece3663DFLIPFLOP
  FF2 (VDD VSS CLK NCLK D2 Q2) ece3663DFLIPFLOP
  FF3 (VDD VSS CLK NCLK D3 Q3) ece3663DFLIPFLOP
  FF4 (VDD VSS CLK NCLK D4 Q4) ece3663DFLIPFLOP
  FF5 (VDD VSS CLK NCLK D5 Q5) ece3663DFLIPFLOP
  FF6 (VDD VSS CLK NCLK D6 Q6) ece3663DFLIPFLOP
  FF7 (VDD VSS CLK NCLK D7 Q7) ece3663DFLIPFLOP
  FF8 (VDD VSS CLK NCLK D8 Q8) ece3663DFLIPFLOP
  FF9 (VDD VSS CLK NCLK D9 Q9) ece3663DFLIPFLOP
  FF10 (VDD VSS CLK NCLK D10 Q10) ece3663DFLIPFLOP
  FF11 (VDD VSS CLK NCLK D11 Q11) ece3663DFLIPFLOP
  FF12 (VDD VSS CLK NCLK D12 Q12) ece3663DFLIPFLOP
  FF13 (VDD VSS CLK NCLK D13 Q13) ece3663DFLIPFLOP
  FF14 (VDD VSS CLK NCLK D14 Q14) ece3663DFLIPFLOP
  FF15 (VDD VSS CLK NCLK D15 Q15) ece3663DFLIPFLOP
ends ece366316B_REGISTER
```

ALU Schematic

Following the same approach laid out on Design Review #1, we used our netlist configurations to create the layout shown below (with Registers at A,B,and OUT). The top wire represents the Pass A connection and is 16 bits wide.



The following netlist was used to realize this design (ignoring subcircuit definitions shown before).

```
//16-Bit OR
OR_BLOCK (vdd! 0 OUT0Q OUT1Q OUT2Q OUT3Q OUT4Q OUT5Q OUT6Q OUT7Q OUT8Q OUT9Q OUT10Q OUT11Q
OUT12Q OUT13Q OUT14Q OUT15Q OUT0W OUT1W OUT2W OUT3W OUT4W OUT5W OUT6W OUT7W OUT8W OUT9W OUT10W
OUT11W OUT12W OUT13W OUT14W OUT15W OUT0_OR OUT1_OR OUT2_OR OUT3_OR OUT4_OR OUT5_OR OUT6_OR
OUT7_OR OUT8_OR OUT9_OR OUT10_OR OUT11_OR OUT12_OR OUT13_OR OUT14_OR OUT15_OR) ece366316B_OR

//16-Bit AND
AND_BLOCK (vdd! 0 OUT0Q OUT1Q OUT2Q OUT3Q OUT4Q OUT5Q OUT6Q OUT7Q OUT8Q OUT9Q OUT10Q OUT11Q
OUT12Q OUT13Q OUT14Q OUT15Q OUT0W OUT1W OUT2W OUT3W OUT4W OUT5W OUT6W OUT7W OUT8W OUT9W OUT10W
OUT11W OUT12W OUT13W OUT14W OUT15W OUT0_AND OUT1_AND OUT2_AND OUT3_AND OUT4_AND OUT5_AND
OUT6_AND OUT7_AND  UT8_AND OUT9_AND OUT10_AND OUT11_AND OUT12_AND OUT13_AND OUT14_AND
OUT15_AND) ece366316B_AND

//16-Bit SHIFTER  [-!-!- Make into SUBCKT -!-!-]


  S0 (vdd! 0 OUT0Q OUT0W B0Bar OUT1W B1Bar O1 O2 O3 O4) oneBitShift
  S1 (vdd! 0 OUT1Q OUT0W B0Bar OUT1W B1Bar O2 O3 O4 O5) oneBitShift
  S2 (vdd! 0 OUT2Q OUT0W B0Bar OUT1W B1Bar O3 O4 O5 O6) oneBitShift
  S3 (vdd! 0 OUT3Q OUT0W B0Bar OUT1W B1Bar O4 O5 O6 O7) oneBitShift
  S4 (vdd! 0 OUT4Q OUT0W B0Bar OUT1W B1Bar O5 O6 O7 O8) oneBitShift
  S5 (vdd! 0 OUT5Q OUT0W B0Bar OUT1W B1Bar O6 O7 O8 O9) oneBitShift
  S6 (vdd! 0 OUT6Q OUT0W B0Bar OUT1W B1Bar O7 O8 O9 O10) oneBitShift
  S7 (vdd! 0 OUT7Q OUT0W B0Bar OUT1W B1Bar O8 O9 O10 O11) oneBitShift
  S8 (vdd! 0 OUT8Q OUT0W B0Bar OUT1W B1Bar O9 O10 O11 O12) oneBitShift
  S9 (vdd! 0 OUT9Q OUT0W B0Bar OUT1W B1Bar O10 O11 O12 O13) oneBitShift
  S10 (vdd! 0 OUT10Q OUT0W B0Bar OUT1W B1Bar O11 O12 O13 O14) oneBitShift
  S11 (vdd! 0 OUT11Q OUT0W B0Bar OUT1W B1Bar O12 O13 O14 O15) oneBitShift
  S12 (vdd! 0 OUT12Q OUT0W B0Bar OUT1W B1Bar O13 O14 O15 O16) oneBitShift
  S13 (vdd! 0 OUT13Q OUT0W B0Bar OUT1W B1Bar O14 O15 O16 O17) oneBitShift
  S14 (vdd! 0 OUT14Q OUT0W B0Bar OUT1W B1Bar O15 O16 O17 O18) oneBitShift
  SO1 (vdd! 0 0 OUT0W B0Bar OUT1W B1Bar nc0 nc1 nc2 O0) oneBitShift
  SO2 (vdd! 0 0 OUT0W B0Bar OUT1W B1Bar nc3 nc4 O0 O1) oneBitShift
  SO3 (vdd! 0 0 OUT0W B0Bar OUT1W B1Bar nc5 O0 O1 O2) oneBitShift
  SO4 (vdd! 0 0 OUT0W B0Bar OUT1W B1Bar O0 O1 O2 O3) oneBitShift

//16-Bit Add/Sub
8bMA1 (vdd! 0 OUT0Q OUT1Q OUT2Q OUT3Q OUT4Q OUT5Q OUT6Q OUT7Q OUT0W OUT1W OUT2W OUT3W OUT4W
OUT5W OUT6W OUT7W S0 S1 S2 S3 S4 S5 S6 S7 Cin n1) MirrorAdder_8bit
8bMA2 (vdd! 0 OUT8Q OUT9Q OUT10Q OUT11Q OUT12Q OUT13Q OUT14Q OUT15Q OUT8W OUT9W OUT10W OUT11W
OUT12W OUT13W OUT14W OUT15W S8 S9 S10 S11 S12 S13 S14 S15 n1 Cout) MirrorAdder_8bit
```

```
//Input Register for A
REG_A (vdd! 0 CLK NCLK IN 0 IN IN IN 0 IN IN IN IN IN IN IN IN IN IN OUT0Q OUT1Q OUT2Q OUT3Q
OUT4Q OUT5Q OUT6Q OUT7Q OUT8Q OUT9Q OUT10Q OUT11Q  OUT12Q OUT13Q OUT14Q OUT15Q)
ece366316B_REGISTER

//Inut Register for B
REG_B ( vdd! 0 CLK NCLK INX IN IN INX INX INX IN IN INX INX INX INX INX INX INX INX OUT0W
OUT1W OUT2W OUT3W OUT4W OUT5W OUT6W OUT7W OUT8W OUT9W OUT10W OUT11W OUT12W OUT13W OUT14W
OUT15W) ece366316B_REGISTER

//Output Register
REG_RESULT (vdd! 0 CLK NCLK OUT_MUX0 OUT_MUX1 OUT_MUX2 OUT_MUX3 OUT_MUX4 OUT_MUX5 OUT_MUX6
OUT_MUX7 OUT_MUX8 OUT_MUX9 OUT_MUX10 OUT_MUX11 OUT_MUX12 OUT_MUX13 OUT_MUX14 OUT_MUX15 OUT0X
OUT1X OUT2X OUT3X OUT4X OUT5X OUT6X OUT7X OUT8X OUT9X OUT10X OUT11X OUT12X OUT13X OUT14X
OUT15X ) ece366316B_REGISTER


//MUX


//16 8:1 Muxes to select output
MUX0 ( vdd! 0 OUT0_OR OUT0_AND O0 S0 S0 OUT0Q 0 0 SEL0 SEL1 SEL2 OUT_MUX0) ece3663_8MUX
MUX1 ( vdd! 0 OUT1_OR OUT1_AND O1 S1 S1 OUT1Q 0 0 SEL0 SEL1 SEL2 OUT_MUX1) ece3663_8MUX
MUX2 ( vdd! 0 OUT2_OR OUT2_AND O2 S2 S2 OUT2Q 0 0 SEL0 SEL1 SEL2 OUT_MUX2) ece3663_8MUX
MUX3 ( vdd! 0 OUT3_OR OUT3_AND O3 S3 S3 OUT3Q 0 0 SEL0 SEL1 SEL2 OUT_MUX3) ece3663_8MUX
MUX4 ( vdd! 0 OUT4_OR OUT4_AND O4 S4 S4 OUT4Q 0 0 SEL0 SEL1 SEL2 OUT_MUX4) ece3663_8MUX
MUX5 ( vdd! 0 OUT5_OR OUT5_AND O5 S5 S5 OUT5Q 0 0 SEL0 SEL1 SEL2 OUT_MUX5) ece3663_8MUX
MUX6 ( vdd! 0 OUT6_OR OUT6_AND O6 S6 S6 OUT6Q 0 0 SEL0 SEL1 SEL2 OUT_MUX6) ece3663_8MUX
MUX7 ( vdd! 0 OUT7_OR OUT7_AND O7 S7 S7 OUT7Q 0 0 SEL0 SEL1 SEL2 OUT_MUX7) ece3663_8MUX
MUX8 ( vdd! 0 OUT8_OR OUT8_AND O8 S8 S8 OUT8Q 0 0 SEL0 SEL1 SEL2 OUT_MUX8) ece3663_8MUX
MUX9 ( vdd! 0 OUT9_OR OUT9_AND O9 S9 S9 OUT9Q 0 0 SEL0 SEL1 SEL2 OUT_MUX9) ece3663_8MUX
MUX10 ( vdd! 0 OUT10_OR OUT10_AND O10 S10 S10 OUT10Q 0 0 SEL0 SEL1 SEL2 OUT_MUX10) ece3663_8MUX
MUX11 ( vdd! 0 OUT11_OR OUT11_AND O11 S11 S11 OUT11Q 0 0 SEL0 SEL1 SEL2 OUT_MUX11) ece3663_8MUX
MUX12 ( vdd! 0 OUT12_OR OUT12_AND O12 S12 S12 OUT12Q 0 0 SEL0 SEL1 SEL2 OUT_MUX12) ece3663_8MUX
MUX13 ( vdd! 0 OUT13_OR OUT13_AND O13 S13 S13 OUT13Q 0 0 SEL0 SEL1 SEL2 OUT_MUX13) ece3663_8MUX
MUX14 ( vdd! 0 OUT14_OR OUT14_AND O14 S14 S14 OUT14Q 0 0 SEL0 SEL1 SEL2 OUT_MUX14) ece3663_8MUX
MUX15 ( vdd! 0 OUT15_OR OUT15_AND O15 S15 S15 OUT15Q 0 0 SEL0 SEL1 SEL2 OUT_MUX15) ece3663_8MUX
```